CS10720 Problems and Solutions

Thomas Jansen

Today: Computability

April 14th

Plans for Today

Towards Our Goal

Remember want to find out what kind of computational problem computers cannot solve

What do we mean by 'cannot solve'?

What we do NOT mean is cannot be solved because of

- lack of resources (memory, time, . . .)
- lack of a clever idea
- we do not quite understand problem

What we DO mean is
cannot be solved for principle reasons
not today, not tomorrow—never

Remember

- computational problem
- reduction
- Turing machine

Computational Problems

Definition (Computational Problem)

A computational problem is defined by a set of finite inputs over a finite input alphabet and for each input a set of correct finite outputs over a finite output alphabet.

Example input directed graph with edge weights, nodes A and B
output shortest path from A to B

Definition (Optimisation Problem)

A optimisation problem is a computational problem where the output is the value of an optimal solution.

Example input directed graph with edge weights, nodes A and B
output length of shortest path from A to B

Definition (Decision Problem)

A decision problem is a computational problem where the output is 'yes' or 'no' (alternatively, '0' or '1' if we prefer binary encodings).

Example input directed graph with edge weights, nodes A and B, value k
output yes if there is a path from A to B of length $\leq k$, no otherwise

Reductions

Fact Solving some problem P
using an algorithm for some other problem Q
is called 'reducing P to Q'
and implies in some sense P is not much harder than Q

Special case solving decision problem P
using an algorithm for some other decision problem Q
is called 'reducing P to Q' and written as $P \leq Q$
and implies in some sense P is not much harder than Q

Observation reductions can help to find
more problems that computers cannot solve
If we know that P cannot be solved by computer
and if we know that $P \leq Q$
then Q cannot be solved by computer
because we can solve P with the help of Q
and the '$\leq$'-algorithm

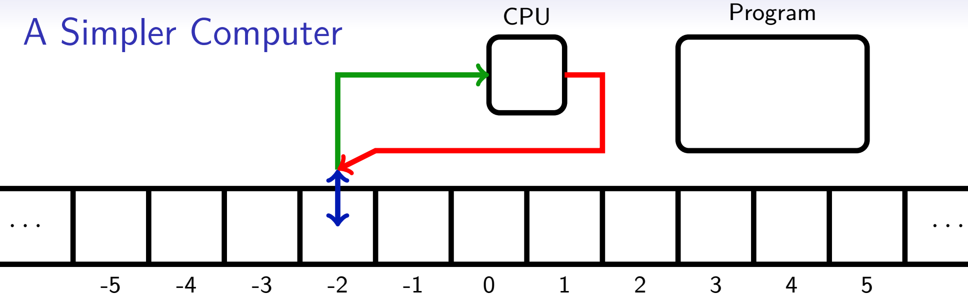
Remember: Turing Machine

Definition (Turing Machine)

A Turing machine has a finite set of possible states Q, an initial state $q_0 \in Q$, a finite memory alphabet Γ that contains the blank character _, a finite input alphabet Σ that does not contain _, an infinite memory that is linearly organised, and a current position in the memory. Initially the input is in the memory, the current position is the first position of the input and all unused memory cells contain _. Its functioning is defined by a program $P\colon Q \times \Gamma \to Q \times \Gamma \times \{L, R, *\}$. It operates in steps, in each step it is in some state $q \in Q$ and reads the contents of the current cell $a \in \Gamma$. If $P(q, a) = (r, b, d)$ then it replaces a by b, changes state from q to r and changes the current cell to its neighbour if $d = L$, to its right neighbour if $d = R$, leaving it unchanged if $d = *$.

Observe Turing machine is our simpler computer
can solve exactly all problems that any computer can solve

A Simpler Computer



- infinite number of memory cells
- CPU that is in a state
- program the CPU follows (by executing it)
- CPU operates in steps (one command of program in one step)
- in each step, CPU can read from current cell of memory
- in each step, CPU can write to the same cell of memory
- in each step, CPU can change its state
- in each step, current cell can change to neighbouring cell

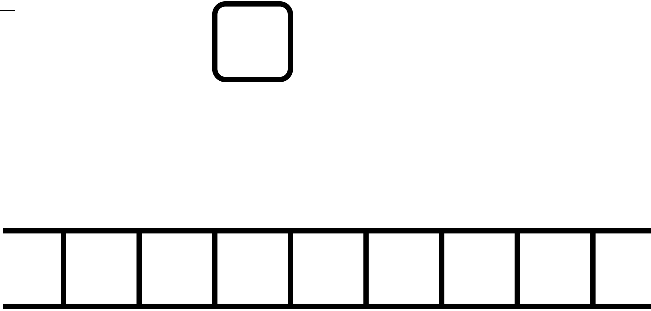
Observation is Turing machine

Example: Decide if Number of 1s in Input is Odd

- input alphabet $\Sigma = \{0, 1\}$
- memory alphabet $\Gamma = \{0, 1, \_\}$
- states $Q = \{q_0, q_1, q_{\mathsf{yes}}, q_{\mathsf{no}}\}$

Program P

$\boldsymbol{(q, a)}$	$\boldsymbol{(r, b, d)}$
$(q_0, 0)$	$(q_0, \_, R)$
$(q_0, 1)$	$(q_1, \_, R)$
$(q_0, \_)$	$(q_{\mathsf{no}}, \_, *)$
$(q_1, 0)$	$(q_1, \_, R)$
$(q_1, 1)$	$(q_0, \_, R)$
$(q_1, \_)$	$(q_{\mathsf{yes}}, \_, *)$



What Do we Mean By ‘Cannot Solve’?

When do we say a computer solves a computational problem?

We expect

- for each input a correct output
- in finite time.

Again to be crystal clear
We say ‘program P solves the computational problem Q’
if for any input that is an instance of Q
P stops eventually and outputs a correct solution
for that instance

Definition (Computability)

We call a computational problem computable if there exists a program that solves it.

Are there computational problems that are not computable?
If so, are any of them practically important?

Concentrating on Decision Problems

Fact we will be concentrating on decision problems
i. e., problems where the output is yes/no (or 1/0)

Is this is fundamental restriction?

Observation not a fundamental restriction ✔
because 1 often reduction of computational problem to decision problem possible
2 always possible to use binary encoding for output of computational problem binary and use separate decision problem for each bit of the output

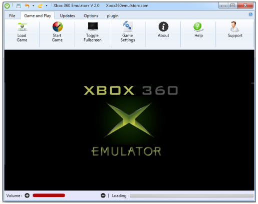
Remember decision problems are computational problems
just restricted ones
(with strong restriction on possible output)

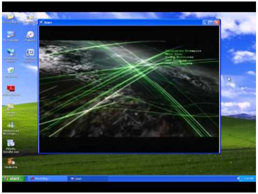
Towards an Important Non-Computable Problem

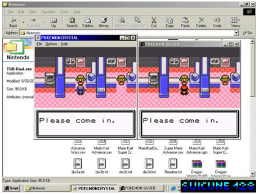
We want to know Is there a non-computable computational problem?
Is there a practically important one?

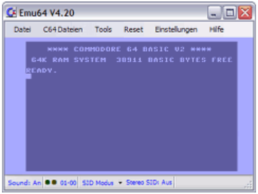
Important concept on the way to an answer
emulation/simulation
'computers pretending to be other computers'
'programs pretending to be other programs'

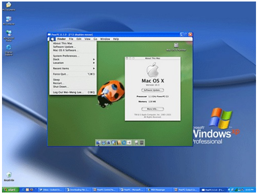
Emulators


Xbox 360 Emulatorx V 2.0
File
Game and Play
Updates
Options
plugin
Load Game
Start Game
Toggle Fullscreen
Game Settings
About
Help
Support
XBOX 360
EMULATOR
Volume :
Loading :




POKEMONCRYSTAL
POKEMON SILVER
Please come in.
Please come in.
Nintendo
SUICUNE 428


Emu64 V4.20
Datei
C64 Dateien
Tools
Reset
Einstellungen
Hilfe
**** COMMODORE 64 BASIC V2 ****
64K RAM SYSTEM 38911 BASIC BYTES FREE
READY.
Sound: An
01-00
SID Modus
Stereo SID: Aus


Mac OS X
Windows XP Professional


NetBeans Feed

Closer to Computational Problems

Observation given a program P and an input I
it is possible for another program to simulate P on I

Is that difficult?

Observation depends on the way P is given
(e. g., C program, Java program, ...)
particularly easy if P is given as a Turing machine
because Turing machines are so restricted

We can simulate any program P on any input I
(the simulation may be slower than running P directly)
(but in the context of computability this is not important)

Correctness of Programs

Why would anyone want to simulate a program P on an input I?

Observation could be useful to test correctness of programs i. e., for program P and input I:

1. Does P stop on I?
2. Is the output P computes for I correct?

Let's concentrate on the partial problem 1

We want program taking as input program P and input I for P and outputs if P stops on I

Observation This is a decision problem.

Definition (Halting problem)

The decision problem that takes as input a program P and an input I for P and that outputs 'yes' if P stops on I and 'no' otherwise is called the halting problem.

Observation halting problem is practically important

Writing a Relatively Simple Program

Let's assume that the halting problem is computable
i. e., we assume program H that solves halting problem exists
i. e., H takes as input a program P and in input I
H stops after some time and outputs 'yes' if P stops on I
and outputs 'no' if P never stops on I

Remark H cannot just simulate P on I
because then H would not stop if P does not stop on I
H somehow needs to recognise if P gets stuck on I
but we do not care how this is done

Remember we assume H solves the halting problem

Constructing a Program

Remember we assume H solves the halting problem

Define program M that accepts as input program P
(format for P is the same as for H)

```
Program M (with input P)
Simulate H on (P,P).
If H outputs 'no' then stop else loop forever.
```

Observations

- M is not particularly difficult to build
- M does not do anything particularly useful
- M stops if H outputs 'no' for P on P as input.
- M does not stop if H outputs 'yes' for P on P as input.

M on a Very Special Input

```
Program M (with input P)
Simulate H on (P,P).
If H outputs 'no' then stop else loop forever.
```

What does M do with M as input?

Observation if M receives a description of M as input
it simulates H on (M, M)

What does H do on input (M, M)?

Observation it stops and outputs either 'yes' or 'no'

Case 1 H outputs 'yes' on $(M, M) \Rightarrow M$ does not stop on input M
because it loops forever if H outputs 'yes'
H was wrong $\Rightarrow$ Case 1 is impossible

Case 2 H outputs 'no' on $(M, M) \Rightarrow M$ stops on input M
because it stops if H outputs 'no'
H was wrong $\Rightarrow$ Case 2 is impossible

In any case H makes a mistake on input (M, M)

The Halting Problem

We see — for program H the supposedly solves the halting problem
we can construct M such that
H makes a mistake on input (M, M)
proving that H does not solve the halting problem

Observation — this works for any program H

Consequence — no program can solve the halting problem
because 'solving' means
stopping on all inputs and deciding correctly

Theorem

The halting problem is not computable.

Summary & Take Home Message

Things to remember

- halting problem
- computability
- halting problem not computable

Take Home Message

- The halting problem is a practical, important problem with significant applications.
- No computer can ever be able to solve the halting problem.
- There are problems that computers cannot solve and we should be aware of this so that we do not waste our time trying.

Lecture feedback `http://onlineted.com`